Theory and Practice of Field-based Motion Coordination in Multiagent Systems

Marco Mamei[†] Franco Zambonelli[†]

[†] Dipartimento di Scienze e Metodi dell'Ingegneria,
Università di Modena e Reggio Emilia,
Via Allegri 13 – 42100 Reggio Emilia, Italy
{mamei.marco,franco.zambonelli}@unimo.it

Abstract

Enabling and managing coordination activities between autonomous, possibly mobile, computing entities in dynamic computing scenarios challenges traditional approaches to distributed application development and software engineering. This paper specifically focuses on the problem of motion coordination, and proposes field-based coordination as a general framework to model and engineer such coordinated behaviors. The key idea in field-based coordination is to have agents' movements driven by computational force fields, generated by the agents themselves and/or by some infrastructure, and propagated across the environment. This paper shows that field-based approaches enable the definition of adaptive and effective motion coordination schemes, which can be modeled and tested by making use of a dynamical systems formalism, and which can be easily implemented either above existing middleware infrastructures or by making use of novel middleware specifically conceived for field-based coordination.

1 Introduction

As computing is becoming pervasive, autonomous computer-based systems are going to be embedded in all everyday objects and in the physical environment. In such a scenario, mobility too, in different forms, will be pervasive. Mobile users, mobile devices, computer-enabled vehicles, as well as mobile software components, define an open and dynamic networked world that will be the stage of near-future software applications.

Traditionally, software applications have always been built by adopting programming paradigms rooted on non-autonomous components coupled at design time by fixed interaction patterns. Although simple in principle, this approach seems inadequate to deal with the dynamism of the above sketched application scenario and it is likely to lead to ever-more brittle and fragile applications.

Only in recent years, the research on software agents has fostered new programming paradigms based on autonomous components (i.e. components with a separated thread of execution and control) interacting to realize an application [3, 7, 20]. In this paper, the term agent can refer not only to mobile software agents situated in a computational environment (e.g., a distributed world of Web resources), but more generically to any autonomous real-world entity with computing and networking capability (e.g., a user carrying on a WiFi PDA, a robot, or a modern car) situated in a physical

environment (e.g., a street or a building). Autonomous interacting components are intrinsically robust. For example, if a component breaks down or goes out of the wireless network, the others can autonomously and dynamically re-organize their interaction patterns to account for such a problem. The key element leading to such robust, scalable and flexible behaviors is coordination: the agents' fundamental capability to decide dynamically on their own actions in the context of their dynamic operational environment. Autonomous components, in fact, must be able to coordinate their activities' patterns to achieve goals, that possibly exceed their capabilities as singles, despite - and possibly taking advantage - of environment dynamics and unexpected situations [18, 28].

Unfortunately, general and widely-applicable approaches to program and manage these autonomous interacting systems are unknown. The main conceptual difficulty is that it is possible to apply a direct engineered control only on the single agents' activities, while the application task is often expressed at the global-ensemble scale [4, 18]. Bridging the gap between single and global activities is not easy, but it is possible: distributed algorithms for autonomous sensor networks have been proposed and successfully verified [8], routing protocols in MANET (in which devices coordinate to let packets flow from sources to destinations) have been already widely used [21]. The problem is still that the above successful approaches are ad-hoc to a specific application domain and it is very difficult to generalize them to other scenarios. There is a great need for general, widely applicable engineering methodologies, middleware and API's to embed, support and control coordination in multi-agent systems [12, 17, 29].

A significant application scenario related to the above general problem is distributed motion coordination. Distributed motion coordination refers to the general problem of orchestrating global patterns of movement in a set of autonomous agents situated in an environment. The goals of enforcing a specific coordinated motion pattern may be various (e.g., letting agents meet together, distribute themselves in an environment according to specific spatial patterns, or simply letting them efficiently move in an environment without interfering with each other) and may have useful applications in a variety of distributed scenarios, from Internet and Grid computing, to traffic control and digital tourism.

The question of how one can effectively model and implement motion coordination patterns in an ensemble of distributed agents has to be evaluated against a number of strict requirements. First, due to both the intrinsic decentralization of the scenario and the autonomy of application components, centralized approaches are ruled out: motion coordination must be enforced in a fully distributed way. Second, due to the intrinsic dynamics and openness of most scenarios of interest, a motion coordination approach should lead to the definition of adaptive strategies, capable of working well independently of the specific characteristics of the environment and of the number and identities of the agents to be coordinated. Third, as is always the case, an approach should not impose too much computational and communication burden on agents.

Unfortunately, traditional coordination models and middleware do not suit the needs of motion coordination in decentralized and dynamic environments. In models and systems relying on message-passing between agents [3], agents typically have to communicate intensively with each other to evaluate the current conditions of the systems (i.e., to become "context-aware") and to negotiate their current movements in the environment on the basis of an a priori encoded strategy. In models and systems relying on common interaction spaces [7], agents can more easily access contextual information about the current state of the system from the shared interaction spaces, but are still left the duty of internally computing such information to implement some a priori coded strategy. The result, in both cases, is a motion coordination strategy that is achieved with high

communication and computation costs, and which tends to be brittle and fragile. The alternative approach we intend to discuss in this paper is field-based coordination [13, 11, 14].

In field-based coordination, agents interact with each other and with the operational environment by simply generating and perceiving distributed data structures abstracting sorts of "gravitational fields". On the one hand, depending on the specific motion pattern to be enforced, different types of fields will be generated by agents and by the environment. On the other hand, agents can locally perceive these fields and decide where to go simply on the basis of the fields they sense and of their magnitude. Eventually, a global adaptive motion pattern emerges in a fully distributed way due to the inter-related effects of agents following the fields' shape and of dynamic fields re-shaping due to agents' movements.

Here we firstly introduce the concept of field-based coordination and discuss related proposals (Section 2). Then, we show how the physical metaphor underlying field-based coordination enables modeling motion coordination problems in terms of a simply dynamical systems formalism (Section 3). Following, we discuss how the general concept of field-based coordination may be practically implemented and programmed by making use either of existing middleware infrastructure or, better, by making use of novel middleware infrastructures specifically conceived to facilitate the programming of complex motion patterns (Section 4). Then, we emphasize on the fundamental role of spatial abstractions in field-based approaches (Section 5). Eventually, some final remarks and conclusions are reported (Section 6).

2 Field-based Approaches to Motion Coordination

Field-based coordination gets inspiration from the physical world, and in particular from the way masses and particles in our universe adaptively move and globally self-organize their movements accordingly to the locally perceived magnitude of gravitational/electro-magnetic fields. The key point is that gravitational fields play the very important role of locally providing agents with a global representation of the current situation of the system (i.e., of the context in which agents are situated) that is immediately usable by agents without further computation and communication activity.

2.1 Key Concepts of Field-based Coordination

Field-based approaches to motion coordination aim at providing agents with abstract - simple yet effective - representations of the context. To this end, field-based approaches delegate to a middleware infrastructure or to the agents themselves (connected in a peer-to-peer ad-hoc network) the task of constructing and automatically updating an essential distributed "view" of the system situation - possibly tailored to application-specific motion coordination problems - that "tells" agents what to do (i.e., how to move to implement a specific motion coordination patterns). Agents are simply left with the decision of whether to follow such a suggestion or not.

Operatively, the key points of the field-approach can be summarized as follows:

1. The environment is represented and abstracted by "computational fields", propagated by agents or by an environmental network infrastructure. These fields convey useful information for the agents' coordination tasks and provide agents with strong coordination-task-tailored context awareness;

2. The motion coordination policy is realized by letting agents move following the gradients of these fields.

3. Environmental dynamics and agents' movements induce changes in the fields' surface, composing a feedback cycle that influences agents' movement (item 2).

4. This feedback cycle let the system (agents, environment and infrastructure) auto-organize, so that the coordination task is eventually achieved in an adaptive and fully distributed way.

A computational field can be considered a sort of spatially distributed data structure characterized by a unique identifier, a location-dependent numeric value, and a propagation law determining how the numeric value should change in space. Fields are locally accessible by agents depending on their location, providing them and a local perspective of the global situation of the system. For instance, consider the personal agent of a museum guide that, in the museum, propagates a simple computational field whose numeric value monotonically increases from room to room as it gets farther from the source agent (See figure 1). Clearly, such field will represent a sort of gravitational field emitted by the agent itself and will enable the personal agents of any museum visitor, from anywhere in the museum, to "sense" where the guide is.

Clearly, the simple principle to enforce field-based motion coordination is having agents move following the local shape of specific fields. For instance, a tourist in a museum looking for the closest guide could simply instruct his personal agent to follow downhill to a local minima the overall gravitational fields emitted by the museum guides. Dynamic changes in the environment and agents' movements induce changes in the fields' surface, producing a feedback cycle that consequently influences agents' movement. For instance, should the museum guide be moving around in the museum, the field-supporting infrastructure would automatically update the corresponding computational field of the guide and would, consequently, have any tourist agent looking for a guide re-adapt his movement accordingly. This concept also makes motion coordination strategy intrinsically adaptive. For instance, should there be multiple guides in the museum, they could decide to sense each other's fields so as to maintain a certain distance from each other to improve their reachability by tourists. If a guide has to go away, the same fields would allows the others to re-shape their formation automatically and adaptively.

In a given environment, the agents as well as the infrastructure can decide to propagate any kind of fields, to facilitate the achievement of application-specific problems. However, the achievement of an application-specific coordination task is rarely relying on the evaluation, as it is, of an existing computational field. Rather, in most cases, an application-specific task relies on the evaluation of an application-specific *coordination field*, as a combination of some of the locally perceived fields. The *coordination field* is a new field in itself, and it is built with the goal of encoding in its shape the agent's coordination task. Once a proper *coordination field* is computed, agents can achieve their coordination task by simply following the shape of their *coordination field*. For instance, for the guides of the museum to stay as far as possible from each other, they simply have to follow uphill a *coordination field* resulting from the sum of all the computational fields of each guide.

2.2 Related Work

Several proposals focusing on field-based approaches for motion coordination have been proposed in the last few years.

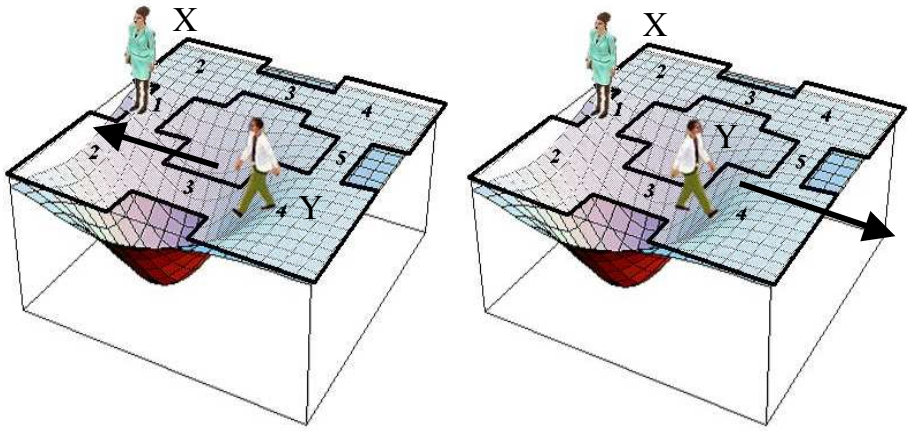

Figure 1: (left) The tourist Y senses the field generated by the tourist guide X (increasing with the distance from the source, as the numeric values in each room indicates). (left) Y agent follows the field downhill to approach X; (right) Y follows the field uphill to get farther from X. In this figure, agent Y uses the field generated by X directly as an application-specific *coordination field*.

The idea of fields driving the collective movements of autonomous robots is by no means new [13]. For instance, one of the most recent re-issue of this idea, the Electric Field Approach (EFA) [11], has been exploited in the control of a team of Sony Aibo four-legged robots in the RoboCup domain. Each Aibo robot builds a field-based representation of the environment from the images captured by its head mounted cameras (stereo-vision), and decides its movements by examining the fields' gradients of this representation. The key point is that, in these approaches, it is up to agents to internally build (at high computational costs) a field-based representation of the context.

Another application scenario in which field-based coordination has been successfully applied is the control of Unmanned Aerospace Vehicles (UAVs) [19]. In this research, fields are implemented by means of distributed data structures spread across a sensor network deployed in the battlefield. Fields can represent targets or enemy forces, and UAVs can fly autonomously by sensing and reacting to fields distribution.

An area in which the problem of achieving effective context-awareness and adaptive coordination has been addressed via a field-based approach is amorphous computing [17]. The particles constituting an amorphous computer have the basic capabilities of propagating hop-by-hop in an ad-hoc way sorts of abstract computational fields in the network, and to sense and react to such fields. Such mechanism can be used, among other possibilities, to drive particles' movements and let the amorphous computer self-assemble in a specific shape.

A very similar approach to self-assembly has been proposed in the area of modular robots [24]. A modular robot is a collection of simple autonomous actuators with few degrees of freedom connected with each other. A distributed control algorithm is executed by all actuators to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait). Currently proposed approaches [24, 26] adopt the biologically inspired idea of hormones to control such a robot. Hormone signals are similar to fields in that they propagate through the network without specific destinations, their content can be modified during propagation, and they may trigger different actions

for different receivers.

Shifting from physical to virtual movements, the popular videogame "The Sims" [25] exploits sorts of computational fields, called "happiness landscapes" and spread in the virtual city in which characters live, to drive the movements of non-player characters. In particular, non-player characters autonomously move in the virtual Sims city with the goal of increasing their happiness by climbing the gradients of specific computational fields (e.g., following the gradient of the fridge field when they are hungry).

3 Modeling Field-based Coordination

The physical inspiration of field-based approaches invites thinking at and modeling field-based coordinated systems in terms of a dynamical system.

3.1 The Dynamical Systems Formalism

In a continuous and unconstrained environment, fields can be represented as continuous functions of space and time, and modeling a motion coordination strategy amounts to writing the differential equations governing the motion of points in space (the agents) driven by the gradient of a specific *coordination field* (as a combination of some fields of the environment). After that, numerically integrating the differential equations of the system provides a very effective way to simulate the system, and it can be regarded as an easy and powerful tool to conduct experiments that can quickly verify that a *coordination field* correctly expresses a coordination task, and tune coefficients.

If we consider the agent *i*, denote its coordinates (for sake of notation simplicity we restrict to the 2-D case) as $(x_i(t), y_i(t))$ and its *coordination field* as $CF_i(x, y, t)$, the differential equations governing *i* behavior are in the form:

$$\begin{cases} \frac{dx_i}{dt} = v\frac{\partial CF_i(x,y,t)}{\partial x}(x,y,t) \\ \frac{dy_i}{dt} = v\frac{\partial CF_i(x,y,t)}{\partial y}(x,y,t) \end{cases}$$

if the motion coordination policy implies that *i* follows the *coordination field* uphill. Changing the sign to the right member of the above equations codes the will of the agent to follow the *coordination field* downhill. In fact, the gradient of the agent's *coordination field*, evaluated towards the spatial coordinates, points to the direction in which the *coordination field* increases. So the agent *i* will follow this gradient or will go in the opposite direction depending on if it wants to follow its *coordination field* uphill or downhill. We indicated with *v* a term that models the agent's speed.

The above modeling can be applied to a number of motion coordination problems, and a simple example is reported in the following of this section. However, it is worth noting that, considering those situations in which agent movements do not occur in an open and continuous space but are constrained by some environmental conditions, the dynamical system description should be made more complex. Modeling motion in constrained environment implies the use in equations of either some artificial force fields that constrain agents to stay within the building plan (an approach that we have successfully experienced) or the adoption of a domain not based on $\Re^n$, but on a more general and complex differentiable manifold. Similar considerations can be applied to model/approximate discrete variations of fields in space.

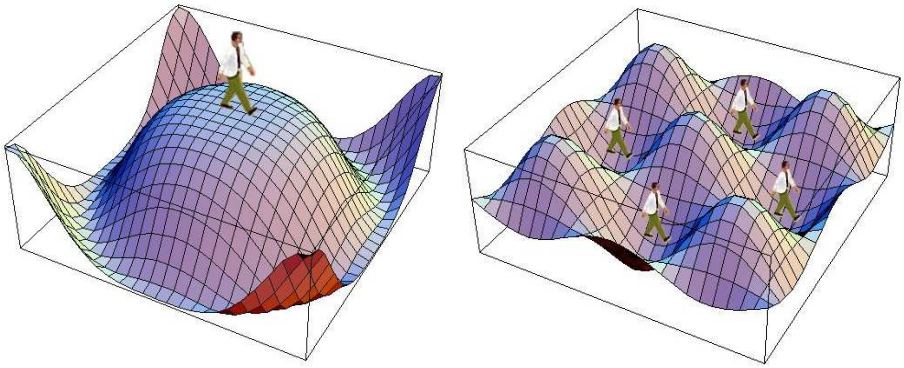

Figure 2: (left) Ideal shape of the flock field. (right) when all the agents follow other agents' fields they collapse in a regular grid formation.

In any case, more than analytical modeling, we think that a proper simulation environment can be effectively exploited to complement the dynamical system description and test the effectiveness of a solution in discrete and constrained environment. For instance, by simply properly programming the Swarm toolkit [27], one could effectively model any required spatial environment (e.g., a city street-plan or a museum map); the presence in such environment of any needed number of fields (possibly changing in space in a discrete way, as it is the case of fields propagated by agents in an ad-hoc way); and the presence of any needed number of agents each with its own goals.

3.2 Modeling Examples

Let us consider the case study of having agents distribute according to specific geometrical patterns ("flocks"), and letting them preserve such patterns while moving. More specifically, agents can represent security guards (or security robots) in charge of cooperatively monitoring a museum by distributing themselves in the museum so as to stay at specified distances from each other [10]. To implement such a coordinated behavior, we can have that each agent generates a field *(flock-field)* (the name deriving from the analogy with the behavior of flocks of birds) whose magnitude assumes the minimal value at specific distance from the source, distance expressing the desired spatial separation between security guards. To coordinate movements, agents have simply to locally perceive the generated tuples, and to follow the gradient of the fields downhill. The expected result is a globally coordinated movement in which agents maintain an almost regular grid formation and preserve this while moving (see figure 2).

Analytically, the *(flock-field)* generated by an agent *i* located at (X_P^i, Y_P^i) can be simply described as follows:

$$d = \sqrt{(x - X_P^i)^2 + (y - Y_P^i)^2}$$

$$FLOCK_i(x, y, t) = d^4 - 2a^2d^2$$

where a is again the distance at which agents must stay away from each other. Starting from these simple equations, one can write, for each of the agents in the set, the differential equations

ruling the dynamic behavior of the system, i.e., expressing that agents follow downhill the minimum of the *(flock-field)*. These have the form:

$$\begin{cases} \frac{dx_i}{dt} = -v\frac{\partial CF_i(x,y,t)}{\partial x}(x_i, y_i) & i = 1, 2..., n \\ \frac{dy_i}{dt} = -v\frac{\partial CF_i(x,y,t)}{\partial y}(x_i, y_i) & i = 1, 2..., n \end{cases}$$

$$\begin{cases} \frac{dx_i}{dt} = -v\frac{\partial min(FLOCK_1, FLOCK_2, ..., FLOCK_n)}{\partial x}(x_i, y_i) & i = 1, 2..., n \\ \frac{dy_i}{dt} = -v\frac{\partial min(FLOCK_1, FLOCK_2, ..., FLOCK_n)}{\partial y}(x_i, y_i) & i = 1, 2..., n \end{cases}$$

Such equations can be numerically integrated by making use of any suitable mathematical software. In our studies, we used the Mathematica package [16]. Figure 3 shows the results obtained by integrating the above equations, for different initial conditions, and for a system composed by four agents. The figure shows a *x-y* plane with the trajectories of the agents of the system (i.e. the solutions of $(x_i(t), y_i(t))$ evaluated for a certain time interval) while moving in a open space. It is rather easy to see that the four agents maintain a formation, trying to maintain specified distances form each other. Figure 4 shows the result of the motion coordination in a simulated environment, realized via the Swarm toolkit [27]. Specifically, a simple museum map has been draw, with three agents (the white dots) moving in it. The formation consists in having the agents remain in adjacent rooms from each other.

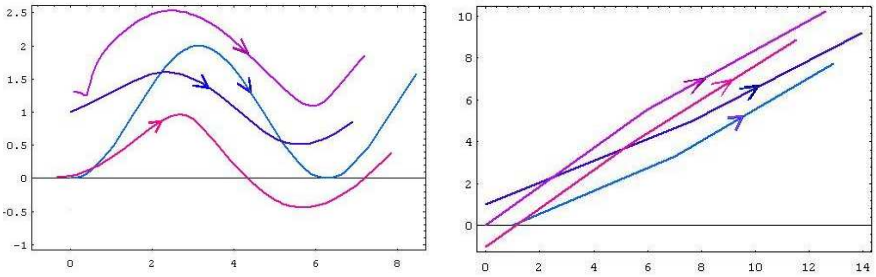

Figure 3: Solutions of the flock fields' differential equations, for different initial conditions.

3.3 The Hint for a Methodology

It is interesting to notice that the equations governing the motion of a Co-Field agent can be also interpreted as partial differential equations prescribing what *coordination field* an agent has to sense in order to move accordingly to a specified pattern.

In theory this is very useful since, from a methodology point of view, it enables answering the question: *What kind of coordination field is required to move following a specific trajectory?* This is a fundamental question, since one of the main problems of field-based approaches is the lack of a methodology to help identifying, given a specific motion pattern to be enforced, which fields have to be defined, how they should be propagated, and how they should be combined in a *coordination field*.

For example, if the coordination problem at hand requires an agent to move along a sinusoidal path:

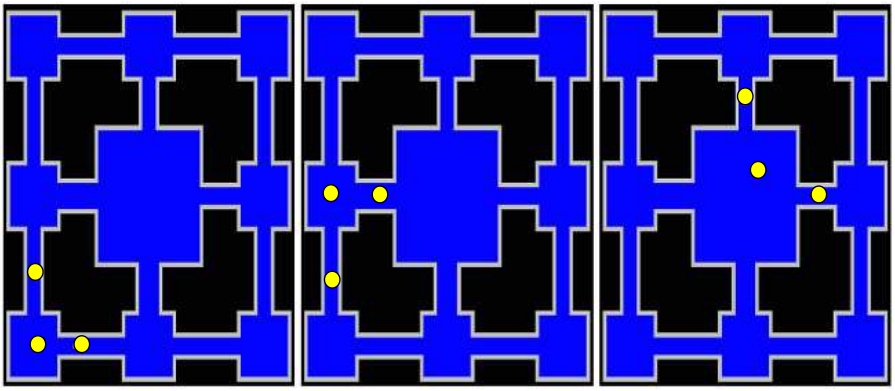

Figure 4: Flocking: from left to right, different stages in the movement of a group of agents through the building and coordinating with each other so as to stay always in neighbor rooms.

$$\begin{cases} x(t) = t \\ y(t) = sin(t) \end{cases}$$

Then, one has basically to find a *coordination field* that solves the following partial differential equations, and let it guide the agent:

$$\begin{cases} v\frac{\partial CF_i(x,y,t)}{\partial x}(x,y,t) = 1 \\ v\frac{\partial CF_i(x,y,t)}{\partial y}(x,y,t) = cos(t) \end{cases}$$

In principle, solving (numerically) the above equations provides a complete representation of the required *coordination field*. It is worth noting that although a closed-form smooth solution may not exist, close-enough solutions with possible discontinuities may do the work (see [28]).

Although the above equations can provide useful hints, they do not solve the methodology problem completely. In fact, one has to consider that the *coordination field* is not something actually spread in the environment by an abstract being. It is computed by the agent by combining fields spread by other agent and/or by some infrastructure. In turn, these fields may change due to the agent movements that are guided by their *coordination field*.

This feedback cycle, that is also the one at the basis of the field-based approach, makes it rather difficult to devise which are the fields spread by the agents and how should they be composed to obtain the *coordination field* satisfying the above equations.

Despite these difficulties, solving the above equations may provide useful hints on which kind of fields to employ to achieve a desired motion pattern.

Complementarily to the above approach, the immediate applicability of field-based protocols is guaranteed by the possibility of getting inspiration from (and of reverse engineering) a wide variety of motion patterns present in nature. Phenomena such as diffusion, birds' flocking, ants' foraging, bee dances [4], to mention few examples, can all be easily modeled with fields (i.e., in terms of agents climbing/descending a *coordination field* obtained as the composition of some computational fields), and all have practical application in mobile computing scenarios.

Finally, it is also important to remark that the field-based approach is not limited to a single coordination field to be followed from the beginning to the end of the application. On the contrary, complex coordination actions can be divided into simpler sub-applications and agents can shift between different field configurations, incrementally realizing complex tasks [17, 26]. This for example can be useful to realize complex non-greedy behaviors (i.e. agent can take local negative actions - climbing a field hill instead descending it - that lead to greater advantages in the future) [19].

4 Supporting Field-based Motion Coordination

Turning the discussion from theory to practice, the basic questions are: What software architecture is required to support a field-based model? How can it be implemented? How can it be programmed? In this section, we first discuss how most modern middleware can be programmed to support a field-based coordination model, then we show how the TOTA middleware, specifically conceived for field-based coordination, can be a much more effective solution.

4.1 Generalities on Implementing Field-based Coordination

Field-based coordination models can potentially be implemented, as an overlay network, on any middleware providing basic support for data storing, communication and event-notification. In fact, what is required from the software infrastructure to implement field-based coordination is to provide simple storage mechanisms (to store field values), communication mechanisms (to propagate field values to neighboring peers), and basic event-notification and subscription mechanisms (to notify agents about changes in field values, to enable agents to select those fields in which they are interested, as well as to promptly update the distributed fields structure to support dynamic changes). For instance, in a preliminary set of experiments, field-based coordination has been implemented above a network of MARS reactive tuple spaces [7]. MARS tuples space have been allocated by IEEE 802.11 access points enabling the communication with WiFi PDA, and have been programmed so as to store fields, to notify agents about local field changes and to propagate fields to neighbor access points.

The choice of implementing a field-based coordination system as an additional layer over an existing interaction middleware, although providing for generality and portability, is not the most efficient solution. First, the mismatch between the mechanisms to be provided in the two layers tends to introduce computational and communication inefficiency. Second, the mismatch between the programming abstractions of the exploited middleware (i.e., its APIs) and those required by field-based coordination would notably complicate the implementation of motion coordination strategies.

4.2 The TOTA Middleware

TOTA (Tuples On The Air) has been explicitly developed within our research group to support field-based coordination models and likes [15].

In TOTA, we propose relying on distributed tuples for representing fields. Tuples are not associated to a specific node (or to a specific data space) of the network. Instead, tuples are injected in the network and can autonomously propagate and diffuse in the network accordingly to a specified pattern. Thus, TOTA tuples form a sort of spatially distributed data structure suitable to represent

fields. To support this idea, TOTA is composed by a peer-to-peer network of possibly mobile nodes, each running a local version of the TOTA middleware. Upon the distributed space identified by the dynamic network of TOTA nodes, each component is capable of locally storing tuples and letting them diffuse through the network. Tuples are injected in the system from a particular node, and spread hop-by-hop accordingly to their propagation rule.

Specifically, in TOTA, fields have been realized by means of distributed tuples

T=(C,P)

characterized by a content *C* and a propagation rule *P*. The content *C* is an ordered set of typed fields representing the information carried on by the tuple. The propagation rule *P* determines how the tuple should be distributed and propagated in the network. This includes determining the "scope" of the tuple (i.e. the distance at which such tuple should be propagated and possibly the spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other tuples in the system. In addition, the propagation rules can determine how tuple's content should change while it is propagated.

The spatial structures induced by tuple propagation must be kept coherent despite network dynamism. To this end, the TOTA middleware supports tuples propagation actively and adaptively: by constantly monitoring the network local topology and the income of new tuples, the middleware automatically re-propagates tuples as soon as appropriate conditions occur. For instance, when new nodes get in touch with a network, TOTA automatically checks the propagation rules of the already stored tuples and eventually propagates the tuples to the new nodes. Similarly, when the topology changes due to nodes' movements, the distributed tuple structure automatically changes to reflect the new topology (see figure 5).

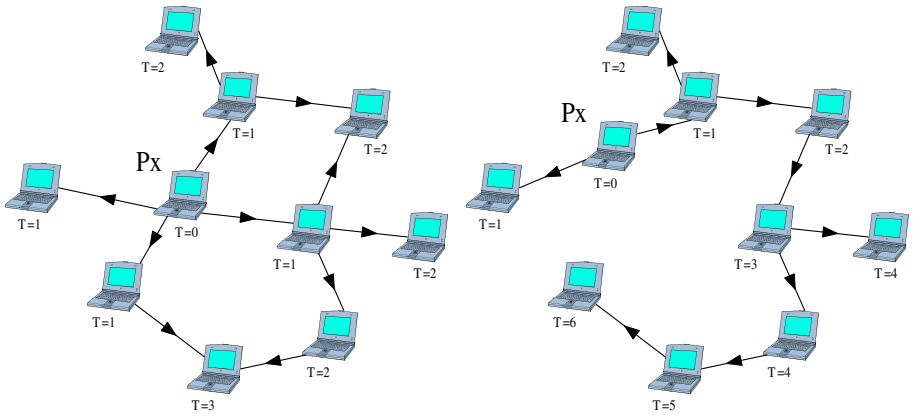

Figure 5: (left) Px propagates a tuple that increases its value by one at every hop. (right) when the tuple source Px moves, all tuples are updated to take into account the new topology

From the architecture point of view, each TOTA middleware has three main parts (see figure 6): *(i)* the TOTA API, is the main interface to access the middleware. It provides functionalities to let the application to inject new tuples in the system (*inject*), to read tuples both from the local tuple space and from the node's one-hop neighborhood, either via pattern matching or via key-access to a tuple's unique id (*read*, *readOneHop*, *keyrd*, *keyrdOneHop*), to delete tuples from the local middleware (*delete*), or to place and remove subscriptions in the event interface (*subscribe*, *unsubscribe*).

Moreover, two methods allow tuples to be stored in the local tuple space or to be propagated to neighboring nodes (*store*, *move*) *(ii)* The EVENT INTERFACE is the component in charge of asynchronously notifying the application about subscribed events. Basically upon a matching event, the middleware invokes on the subscribed components a *react* method to handle the event. *(iii)* The TOTA ENGINE is the core of TOTA: it is in charge of maintaining the TOTA network by storing the references to neighboring nodes and to manage tuples' propagation by executing their propagation methods and by sending and receiving tuples. Within this component is located the tuple space in which to store TOTA tuples.

From an implementation point of view, we developed a first prototype of TOTA running on Laptops and on Compaq IPAQs equipped with 802.11b, Familiar LINUX and J2ME-CDC (Personal Profile). IPAQs connect locally in the MANET mode (i.e. without requiring access points) creating the skeleton of the TOTA network. Tuples are propagated through multicast sockets to all the nodes in the one-hop neighbor. The use of multicast sockets has been chosen to improve the communication speed by avoiding 802.11b unicast handshake. By considering the way in which tuples are propagated, TOTA is very well suited for this kind of broadcast communication. We think that this is a very important feature, because it will allow in the future implementing TOTA also on really simple devices (e.g. micro sensors) that cannot be provided with sophisticated (unicast enabled) communication mechanisms. Other than this communication mechanism, at the core of the TOTA middleware there is a simple event-based engine that monitors network reconfigurations and the income of new tuples and react either by triggering propagation of already stored tuples or by generating an event directed to the event interface.

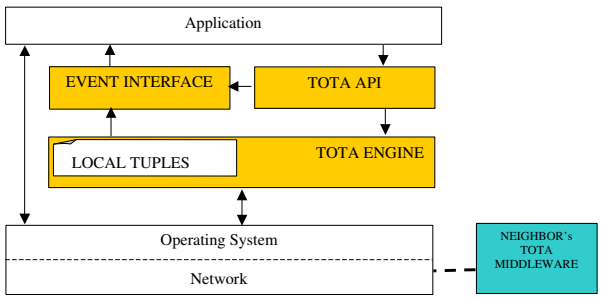

Figure 6: TOTA middleware architecture

4.3 Programming Motion Coordination in TOTA

Relying on an object oriented methodology, TOTA tuples are actually objects: the object state models the tuple content, while the tuple's propagation has been encoded by means of a specific *propagate* method.

When a tuple is injected in the network, it receives a reference to the local instance of the TOTA middleware (via an *init* method). Then its code is actually executed (the middleware invokes the tuple's *propagate* method) and if during execution it invokes the middleware *move* method, the tuple is sent to all the one-hop neighbors, where it will be executed recursively. During migration, the

```
abstract class TotaTuple {
 protected TotaInterface tota;
 /* the state is the tuple content */
 /* this method inits the tuple, by giving a reference
 to the current TOTA middleware */
 public void init(TotaInterface tota) {
  this.tota = tota;
 }
 /* this method codes the tuple actual actions */
 public abstract void propagate();
 /* this method enables the tuple to react
 to happening events see later in the article */
 public void react(String reaction, String event)
{}}
```

Figure 7: The main structure of the *TotaTuple* class

object state (i.e. tuple content) is properly serialized to be preserved and rebuilt upon the arrival in the new host.

Following this schema, we have defined an abstract class *TotaTuple*, that provides a general framework for tuples (i.e., for fields) programming (see figure 7).

In TOTA, a tuple does not own a thread, but it is executed by the middleware (i.e. TOTA ENGINE) that runs the tuple's *init* and *propagate* methods. Tuples, however, must remain active even after the middleware has run their code. This is fundamental because their self-maintenance algorithm must be executed whenever the right condition appears (e.g. when a new peer connects to the network, the tuples must propagate to this newly arrived peer). To this end, tuples can place subscriptions, to the TOTA event interface as provided by the standard TOTA API. These subscriptions let the tuples remain "alive", being able to execute upon triggering conditions.

A programmer can create new tuples by subclassing the *TotaTuple* class. However, to facilitate this task, we developed a tuples' class hierarchy from which the programmer can inherit to create custom tuples. Classes in the hierarchy take care of dealing with propagation and maintenance with regard to a vast number of circumstances.

The class *HopTuple* inherits from *TotaTuple*. This class is a template to create self-maintained distributed data structures over the network. Specifically, it implements the superclass method propagate, as shown in figure 8.

The class *HopTuple* is provided with an integer variable *hop* specifying the hop distance form the source. It implements the methods: *decideEnter*, *decidePropagate*, *changeTupleContent* and *makeSubscriptions* so as to realize a breadth first, expanding ring propagation. The result is simply a tuple that floods the network increasing its *hop* value as it propagates and that maintains such *hop* value despite network topology changes:

- When a tuple arrives in a node (either because it has been injected or it has been sent from a neighbor node) the middleware executes the *decideEnter* method that returns true if the tuple can enter the middleware and actually execute there, false otherwise. The standard implementation returns true if in the node there is not the tuple yet and also if there is the tuple

```
public final void propagate() {
 if(decideEnter()) {
  boolean prop = decidePropagate();
  changeTupleContent();
  this.makeSubscriptions();
  tota.store(this);
  if(prop)
   tota.move(this);
}}
```

Figure 8: Propagate method in the *HopTuple* class

with an higher hop-counter. This allows to enforce the breadth-first propagation assuring that the hop-counter truly reflects the hop distance from the source

- If the tuple is allowed to enter the method *decidePropagate* is run. It returns true if the tuple has to be further propagated, false otherwise. The standard implementation of this method returns always true, realizing a tuple's that floods the network being recursively propagated to all the peers.

- The method *changeTupleContent* changes the content of the tuple. The standard implementation of this method increases an integer hop counter by one at every hop.

- The method *makeSubscriptions* allows the tuple to place subscriptions in the TOTA middleware. As stated before, in this way the tuple can react to events even when they happen after the tuple completes its execution. The standard implementation subscribes to network reconfiguration to implement the self-maintenance algorithm.

- After that, the tuple is inserted in the TOTA tuple space by executing *tota.store(this)*. Again, without this method the tuple would propagate across the network without leaving anything behind. Thus no distributed data structure would be ever being formed.

- Then, if the *decidePropagate* method returned true, the tuple is propagated to all the neighbors via the command *tota.move(this)*. The tuple will eventually reach neighboring nodes, where it will be executed again. It is worth noting that the tuple will arrive in the neighboring nodes with the content changed by the last run of the *changeTupleContent* method.

Programming a TOTA tuple to create a distributed data structure basically reduces to inherit from the above class to overload the four methods specified above to customize the tuple behavior.

A *Gradient* tuple creates a tuple that floods the network in a breadth-first way and have an integer hop-counter that is incremented by one at every hop (see figure 9). Similarly a *FlockingTuple* creates a distributed data structure representing the *flock field* represented in figure 2. In the example the tuple's value decreases in the first two hops, then increases monotonically (see figure 10).

It is rather easy now to program the agent required in the flock case study. With this regard, the algorithm followed by a *FlockingAgent* is very simple: agents have to determine the closest agent, and then move by following downhill that agent's *FlockingTuple*. In this way agents will

```
public class Gradient extends HopTuple {
 public String name;
 public int val = 0;
 protected void changeTupleContent() {
 /* The correct hop value is maintained
 in the superclass HopBasedTuple */
 super.changeTupleContent();
 val = hop;
}}
```

Figure 9: Gradient tuple

```
public class FlockingTuple extends HopTuple {
 /* this is the intended spatial separation
 between agents */
 private static final int RANGE = 2;
 public int value = RANGE;
 protected void changeTupleContent() {
  super.changeTupleContent();
  if(hop <= RANGE)
   value --;
  else
   value ++;
}}
```

Figure 10: Flocking tuple

```
public class FlockingAgent extends Thread
                            implements AgentInterface {
 private TotaMiddleware tota;
 public void run() {
  // inject the flocking tuple to participate the flock
  FlockingTuple ft = new FockingTuple ();
  ft.setContent(peer.toString());
  tota.inject(ft);
  while(true) {
   // read other agents' flocking tuples
   FlockingTuple query = new FlockingTuple();
   Vector v = tota.read(query);
   /* evaluate the gradients and select the peer to
   which the gradient goes downhill */
   GenPoint destination = getDestination(v);
   // move downhill following the meeting tuple
   peer.move(destination);
}}}
```

Figure 11: Flocking agent

collapse in the grid formation as shown in figure 2(right). At start-up, each agent will propagate its *FlockingTuple*. Then it will read its local tuple space to determine the closest agent (the one whose *FlockingTuple* is lower). Then it will inspect its one-hop neighborhood to find the local shape of the *FlockingTuple* of the closest agent. Finally, it will move by following the tuple's gradient downhill (see figure 11).

5 Back to Theory: the Concept of Space in Field-based Approaches.

The type of context-awareness promoted by field-based approaches in general, and by TOTA in particular, is strictly related to spatial-awareness. In fact, by creating distributed data structures over a network, fields intrinsically provide a notion of space in the network. For instance, a field incrementing its content as it propagates hop-by-hop identifies a sort of "structure of space" defining the network distances from the source. This kind of structure of space provides context/spatial awareness to application agents. For example, an agent perceiving the above field with a hop-value X could establish that the source is X hops faraway. Moreover, by inspecting the local slope of the field, the agent could infer the approximate direction to the source (i.e. the direction along which the field goes downhill direction).

Fields, however, also allow dealing with spatial concepts in a much more flexible way. Although at the primitive level the space is the network space and distances are measured in terms of hops between nodes, it is possible to exploit a much more physically-grounded concept of space. This may be required by several application scenarios in which application agents need to interact with and acquire awareness of the physical space. For instance, one can bound the propagation of a field to a portion of physical space by having the propagation procedure - as the field propagates from

node to node - to check the local spatial coordinates, so as to decide whether to propagate the field further or not. In order to bound agents' and fields' behavior to the physical space, nodes must be provided with some kind of localization mechanism [9].

It is interesting to report that a similar idea has been developed and exploited in the context of a recently proposed language to program a vast number of devices dispersed in an environment [6]. The idea of this programming language is to identify a number of spatial regions relevant for a given application and to access the devices through the mediation of these regions (e.g. for all the devices on the "hill" do that). In [6], the definition of the regions is performed adopting GPS devices and distributed data structures similar to fields [5].

Other than the network and the physical space, one could think at mapping the components of a distributed application in any sort of virtual space. This space must be supported by an appropriate routing mechanism allowing distant peers to be neighbors in the virtual space. Such virtual spaces are particularly useful to enable the definition of advanced application such as content-based routing, as in CAN [22] and Pastry [23]. Moreover, virtual spaces extend the applicability of the field-based approaches to application not related to physical movements. In these cases, a coordination field can be thought as spread in an abstract coordination space to encode any kind of coordination actions.

Similar principles have been used in the Multilayered Multi Agent Situated System (MMASS) model [1]. In MMASS, agents' actions take place in a multilayered environment. Each layer provides agents with some contextual information supporting agents' activities. MMASS agents can coordinate with each other by spreading fields across different spaces. An agent will perform actions on the basis of the locally perceived field configuration. The actions performed change the agent's location in the the virtual spaces and thus also the fields the agent perceives and eventually its behavior. For example, in a task-assignment application, MMASS agents could be embedded in a virtual space representing how busy they are. The farther they are from the origin of the space, the more they are busy. When a new task has to be assigned, a field - whose magnitude decays as it is propagated - is injected at the origin of the space. Agents closer to the origin, thus less busy, will perceive the field with greater intensity and they will be more likely to take on the task. As an agent engage a task, it moves in the virtual space to a "more busy" location. This is just an explanatory example and more complex applications can be realized with more fields and more contextual spaces. In our opinion, TOTA and MMASS are almost complementary models. TOTA could effectively serve as the engine on which to realize the abstractions and mechanisms promoted by MMASS [2].

6 Conclusions

In this paper we have presented the concept of field-based coordination, and have discussed both how it can be effectively modeled via dynamical systems formalism and how it can be implemented by exploiting a proper middleware.

Our future work will be twofold. On the one hand, we want to apply field-based coordination to application scenarios that do not deal with physical movements. In these cases, fields can be thought as spread in an abstract coordination space to encode any kind of coordination activities. On the other hand, we would like to define complete engineering methodologies to support the applicability of field-based concepts.

7 Acknowledgements

Work supported by the Italian MIUR and CNR in the "Progetto Strategico IS-MANET, Infrastructures for Mobile ad-hoc Networks".

References

[1] S. Bandini, S. Manzoni, C. Simone, "Heterogeneous Agents Situated in Heterogeneous Spaces", 3rd International Symposium From Agent Theories to Agent Implementations, Vienna (A), April 2002.

[2] S. Bandini, S. Manzoni, G. Vizzari, "Towards a Specification and Execution Environment for Simulations based on MMASS: Managing at-a-distance Interaction", In this volume, 2004.

[3] F. Bellifemine, A. Poggi, G. Rimassa, "JADE - A FIPA2000 Compliant Agent Development Environment", 5th International Conference on Autonomous Agents (Agents 2001), Montreal (CA), May 2001.

[4] E. Bonabeau, M. Dorigo, G. Theraulaz, "Swarm Intelligence", Oxford University Press, 1999.

[5] C. Borcea, D. Iyer, P. Kang, A. Saxena, and L. Iftode. Cooperative computing for distributed embedded systems. In *22nd International Conference on Distributed Computing Systems*. IEEE CS Press, Vienna, Austria, July 2002.

[6] C. Borcea, D. Iyer, P. Kang, A. Saxena, and L. Iftode. Spatial programming using smart messages: Design and implementation. In *24th International Conference on Distributed Computing Systems*. IEEE CS Press, Tokio, Japan, July 2004.

[7] G. Cabri, L. Leonardi, F. Zambonelli, "Engineering Mobile Agent Applications via Context-dependent Coordination", IEEE Transaction on Software Engineering, 28(11):1040-1056, Nov. 2002.

[8] D. Estrin, D. Culler, K. Pister, G. Sukjatme, "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, 1(1):59-69, Jan. 2002.

[9] J. Hightower, G. Borriello, "Location Systems for Ubiquitous Computing", IEEE Computer, 34(8): 57-66, Aug. 2001.

[10] A. Howard, M. Mataric, G. Sukhatme, "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks", Autonomous Robots,special issue on Intelligent Embedded Systems, G. Sukhatme, ed., 2002, 113-126.

[11] S. Johansson, A. Saffiotti, "Using the Electric Field Approach in the RoboCup Domain", Proceedings of the Int. RoboCup Symposium. Seattle, WA, 2001.

[12] J. Kephart, D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, 36(1):41-50, Jan. 2003.

[13] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots", The International Journal of Robotics Research, 5(1):90-98, 1986.

[14] M. Mamei, F. Zambonelli, L. Leonardi, "Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination". IEEE Pervasive Computing, 3(2):52-61, 2004.

[15] M. Mamei, F. Zambonelli, "Self-mantained Distributed Tuples for Field-based Coordination in Dynamic Networks", Proceedings of the SAC Symposium on Applied Computing, ACM Press, Cyprus (CY), March 2004.

[16] Mathematica, http://www.wolfram.com.

[17] R. Nagpal, A. Kondacs, C. Chang, "Programming Methodology for Biologically-Inspired Self-Assembling Systems", in the AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality, March 2003

[18] H. V. Parunak, S. Brueckner, J Sauter, "ERIM's Approach to Fine-Grained Agents", NASA Workshop on Radical Agent Concepts, Greenbelt, MD, USA, Jan. 2002.

[19] H. V. D. Parunak, M. Purcell, R. O'Connell. "Digital Pheromones for Autonomous Coordination of Swarming UAV's". In Proceedings of First AIAA Unmanned Aerospace Vehicles, Systems,Technologies, and Operations Conference, Norfolk, VA, AIAA, 2002

[20] G. P. Picco, A. L. Murphy, G. C. Roman, "LIME: a Middleware for Logical and Physical Mobility", 21st International Conference on Distributed Computing Systems, IEEE CS Press, pp. 524-536, July 2001.

[21] Poor, "Embedded Networks: Pervasive, Low-Power, Wireless Connectivity", PhD Thesis, MIT, 2001.

[22] S. Ratsanamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. In *ACM SIGCOMM Conference*. ACM Press, San Diego (CA), August 2001.

[23] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *18th IFIP/ACM Conference on Distributed Systems Platforms*. ACM Press, Heidelberg (D), November 2001.

[24] W. Shen, B. Salemi, P. Will, "Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots", IEEE Transactions on Robotics and Automation 18(5):1-12, Oct. 2002.

[25] The Sims, http://thesims.ea.com.

[26] K. Stoy and R. Nagpal. Self-reconfiguration using directed growth. In *7th International Symposium on Distributed Autonomous Robotic Systems*. Toulouse, France, June 2004.

[27] The Swarm Simulation Toolkit, http://www.swarm.org.

[28] F. Zambonelli, M. Mamei, "The Cloak of Invisibility: Challenges and Applications", IEEE Pervasive Computing, 1(4):62-70, Oct.-Dec. 2002.

[29] F. Zambonelli, V. Parunak, "From Design to Intentions: Sign of a Revolution", International Conference on Autonomous Agents and Multi-agent Systems, Bologna (I), July 2002.